Kafka Módulo I

Igor Maldonado Floôr

Líder de desenvolvimento @Monitora

Cronograma

Dia	Objetivo
1 – 27/07	Conceitos iniciais de Kafka
2 – 29/07	Conceitos de funcionamento do Kafka
3 – 03/08	Hands on: setup inicial + produtores em java
4 – 05/08	Hands on: Consumidores em java

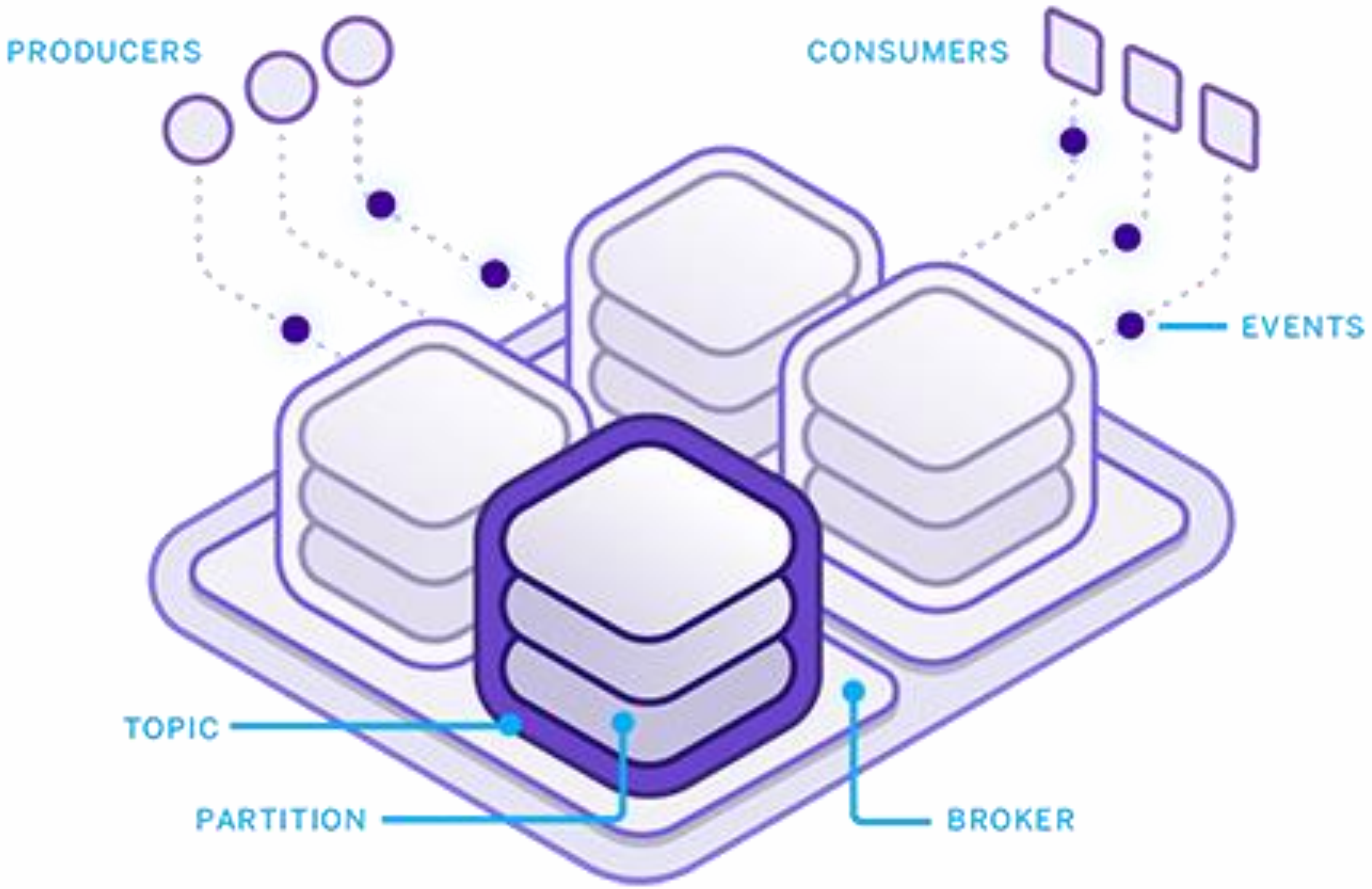
Hands on

Setup

Ferramentas

- IntelliJ Community edition ✓
- Java 12/13 na máquina ✓
- Docker ✓
- Docker compose ✓
  - docker-compose up

https://bit.ly/2Bv2enQ


PRODUCERS
CONSUMERS
EVENTS
TOPIC
PARTITION
BROKER

Cenario live

Docker?

- Docker engine
- Docker desktop

```yaml
version: "2"
services:
  landoop:
    container_name: landoop
    image: landoop/fast-data-dev
    ports:
      - "9092:9092"
      - "2181:2181"
      - "8081:8081"
      - "3030:3030"
      - "9581-9585:9581-9585"
      - "8082:8082"
      - "8083:8083"
    environment:
      - ADV_HOST=127.0.0.1
      - RUNTESTS=0
      - MAX_BYTES=50000000
  portainer:
    container_name: portainer
    image: portainer/portainer:latest
    restart: always
    ports:
      - "9000:9000"
    command: --admin-password '$$2y$$05$$W3R2dlwoQHZlwRxgdwjZp.IBid5M1NgQv0wwHRmLk/pbu4o2xnTMm'
    volumes:
      - "/var/run/docker.sock:/var/run/docker.sock"
```

$git/docker-compose.yml

Desenvolvimento produtor

Passos

1. Projeto inicial
2. Instalar dependências (maven)
3. Criação do model Selling (avro)
4. Configurações de propriedades (properties.yml)
5. Implementação Kafka template + configs
6. Desenvolvimento do produtor

Step 1 - Projeto inicial

- Branch: m1-s1
- $GIT/handsOn/vendedor
- Executar projeto vazio

Step 1 - Projeto inicial

File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help vendedor [C:\Users\Igor\Development\Courses\KafkaCourse\handsOn\vendedor] - IntelliJ IDEA

vendedor

VendedorApplication

Project

vendedor C:\Users\Igor\Development\Courses\KafkaCourse\handsOn\vendedor

- .idea
- .mvn
- src
  - main
    - java
      - com.useargo.vendedor
        - VendedorApplication
    - resources
  - test
- target
- .gitignore
- HELP.md
- mvnw
- mvnw.cmd
- pom.xml
- vendedor.iml
- External Libraries
- Scratches and Consoles

Search Everywhere Double Shift

Go to File Ctrl+Shift+N

Recent Files Ctrl+E

Navigation Bar Alt+Home

Drop files here to open

Run: VendedorApplication

Console Endpoints

```
  \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
   '  |____| .__|_| |_|_| |_\__, | / / / /
  =========|_|==============|___/=/_/_/_/
  :: Spring Boot ::        (v2.3.2.RELEASE)

2020-08-03 01:37:55.585  INFO 15748 --- [           main] c.useargo.vendedor.VendedorApplication   : Starting VendedorApplication on Parindows with PID 15748
 (C:\Users\Igor\Development\Courses\KafkaCourse\handsOn\vendedor\target\classes started by Igor in C:\Users\Igor\Development\Courses\KafkaCourse\handsOn\vendedor)
2020-08-03 01:37:55.586  INFO 15748 --- [           main] c.useargo.vendedor.VendedorApplication   : No active profile set, falling back to default profiles: default
2020-08-03 01:37:55.833  INFO 15748 --- [           main] c.useargo.vendedor.VendedorApplication   : Started VendedorApplication in 0.426 seconds (JVM running for 0.894)

Process finished with exit code 0
```

4: Run 9: Git Terminal GraphQL Build Spring 6: TODO Event Log m1-s1

Step 2 - Instalar dependências

- Trabalhar no arquivo pom.xml
- Ele está na raíz do projeto

Step 2 - Instalar dependências

```
<properties>
    <java.version>12</java.version>
</properties>

<repositories>
    <repository>
        <id>confluent</id>
        <url>http://packages.confluent.io/maven/</url>
        <releases>
            <enabled>true</enabled>
        </releases>
    </repository>
</repositories>

<dependencies>
```

Step 2 - Instalar dependências

```
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter</artifactId>
    </dependency>

    <dependency>
        <groupId>org.springframework.kafka</groupId>
        <artifactId>spring-kafka</artifactId>
    </dependency>
    <dependency>
        <groupId>io.confluent</groupId>
        <artifactId>kafka-avro-serializer</artifactId>
        <version>5.5.0</version>
    </dependency>

    <dependency>
        <groupId>org.apache.avro</groupId>
        <artifactId>avro</artifactId>
        <version>1.9.2</version>
    </dependency>

    <dependency>
```

Step 2 - Instalar dependências

```
    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
            <plugin>
                <groupId>org.apache.avro</groupId>
                <artifactId>avro-maven-plugin</artifactId>
                <version>1.9.2</version>
                <executions>
                    <execution>
                        <phase>generate-sources</phase>
                        <goals>
                            <goal>schema</goal>
                        </goals>
                        <configuration>
                            <sourceDirectory>${project.basedir}/src/main/resources/avro-sources/</sourceDi
rectory>
                            <outputDirectory>${project.basedir}/src/main/java/br/com/monitoratec/vendedor/
</outputDirectory>
                            <stringType>String</stringType>

                        </configuration>
                    </execution>
                </executions>
                <configuration>
                    <imports>
                        <!-- Common -->
<!--                        <import>${project.basedir}/src/main/resources/avro-sources/common/commonRespon
seDTO.avsc</import>-->
                    </imports>
                </configuration>
            </plugin>
```

Step 2 - Instalar dependências

- Instalar dependências pelo maven
- Branch m1-s2

Step 3 - Criação do model Selling (avro)

- Pasta de resources: avro-sources
- Arquivo: selling.avsc

Step 3 - Criação do model Selling (avro)

selling.avsc

```
{
    "type": "record",
    "name": "Selling",
    "namespace": "kafka.avro.generated",
    "fields": [
        {
            "name": "amount",
            "type": "double"
        },
        {
            "name": "buyer",
            "type": "string"
        }
    ]
}
```

Step 3 - Criação do model Selling (avro)

- Executar: mvn package (pode ser pela UI)
- Verificar pasta: sources: kafka/avro/generated
- Branch m1-s3

Step 4 - Configurações de propriedades

- Configuração de propriedades da aplicação (Spring app)

application.yml

```yaml
server:
  port: 5050

spring:
  kafka:
    consumer:
      auto-offset-reset: earliest
      group-id: vendedor
      bootstrap-servers: localhost:9092
    listener:
      missing-topics-fatal: true
    producer:
      bootstrap-servers: localhost:9092

kafka:
  schema-registry-server: http://localhost:8081
```

Step 4 - Configurações de propriedades

- Arquivo está localizado em: resources: applicationyml
- Branch m1-s4

Step 5 - Implementação Kafka template + configs

- Implementação do objeto utilizado para produzir dados no Kafka
- Pasta: Kafka/config

Step 5 - Implementação Kafka template + configs

KafkaProducer.java

```java
package br.com.monitoratec.vendedor.kafka.config;

import ...

@Configuration
public class KafkaProducer {
    private static final Logger LOGGER = LogManager.getLogger(KafkaProducer.class);

    @Value("${spring.kafka.producer.bootstrap-servers}")
    private String bootstrapServers;

    @Value("${kafka.schema-registry-server}")
    private String schemaRegistryServer;

    private Map<String, Object> producerConfigs;

    @PostConstruct
    public void init() {
        this.producerConfigs = new HashMap<>();
        producerConfigs.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, bootstrapServers);
        producerConfigs.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG, StringSerializer.class);
        producerConfigs.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG, KafkaAvroSerializer.class);
        producerConfigs.put(SCHEMA_REGISTRY_URL_CONFIG, schemaRegistryServer);
        LOGGER.warn( s: "Kafka producer configs: {}", producerConfigs);
    }

    @Bean
    public KafkaTemplate<String, Selling> sellingKafkaTemplate() {
        KafkaTemplate<String, Selling> kafkaTemplate = new KafkaTemplate<>(new DefaultKafkaProducerFactory<>(this.producerConfigs));
        kafkaTemplate.setDefaultTopic("br.com.monitoratec.selling");
        return kafkaTemplate;
    }
}
```

Step 5 - Implementação Kafka template + configs

- Branch m1-s5

Step 6 - Desenvolvimento do produtor

- Utilizaremos o objeto Kafka template para escrever no Kafka
- Adição do Seller: sources: seller/Seller.java

Step 6 - Desenvolvimento do produtor

KafkaProducer.java | Seller.java

```java
package br.com.monitoratec.vendedor.seller;

import ...

@Component
public class Seller {

    @Autowired
    KafkaTemplate<String, Selling> sellingKafkaTemplate;

    @EventListener(ApplicationReadyEvent.class)
    public void doSomethingAfterStartup() {
        try {
            Thread.sleep( millis: 5000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        Scanner scanner = new Scanner(System.in);

        String userInput = "";
        while(!userInput.equals("Goodbye")) {
            System.out.println("To do a sell, let the data in the pattern: \n<buyer name>, amount");
            System.out.println("If you want to quit, type: Goodbye");

            userInput = scanner.nextLine();

            if (!userInput.equals("Goodbye")) {
                this.sell(userInput);
            }
        }

    }

    private void sell(String userInput) {
```

Step 6 - Desenvolvimento do produtor

```java
    private void sell(String userInput) {
        if (userInput == null) {
            System.out.println("Impossible to understand, please follow the pattern");
            return;
        }

        String[] pieces = userInput.split( regex: ",");
        if (pieces.length != 2) {
            System.out.println("Impossible to understand, please follow the pattern");
            return;
        }

        String buyer = pieces[0];
        double amount = 0;
        try {
            amount = Double.parseDouble(pieces[1]);
        } catch (Exception e) {
            System.out.println("Impossible to parse the amount, please follow the pattern: 00.00");
            return;
        }

        Selling selling = new Selling();
        selling.setBuyer(buyer);
        selling.setAmount(amount);

        try {
            sellingKafkaTemplate.sendDefault(selling).completable().get();
            System.out.println("Wrote data to broker =]");
        } catch (InterruptedException e) {
            e.printStackTrace();
        } catch (ExecutionException e) {
            e.printStackTrace();
        }

    }
}
```

Step 6 - Desenvolvimento do produtor

- Branch m1-s6

Post hands on

Post hands on

- Verificar mensagens no Broker (via Kafka ui)
- localhost:3030

Dúvidas?

monitora

Software development partner

Passeio São Carlos - Av. Dr. Francisco Pereira Lopes, 1701, Lojas 17 e 18
Parque Santa Mônica, São Carlos/SP - CEP: 13.564-002

(16) 3419-7100 / (16) 3419-7200
www.monitoratec.com.br